Rapid re-meshing and re-solution of three-dimensional boundary element problems for interactive stress analysis

T.M. Foster*, M.S. Mohamed, J. Trevelyan, G. Coates

School of Engineering and Computing Sciences, Durham University, Durham, DH1 3LE, UK

Abstract

Structural design of mechanical components is an iterative process that involves multiple stress analysis runs; this can be time consuming and expensive. It is becoming increasingly possible to make significant improvements in the efficiency of this process by increasing the level of interactivity. One approach is through real-time re-analysis of models with continuously updating geometry. A key part of such a strategy is the ability to accommodate changes in geometry with minimal perturbation to an existing mesh. This work introduces a new re-meshing algorithm that can generate and update a boundary element mesh in real-time as a series of small changes are sequentially applied to the associated model. The algorithm is designed to make minimal updates to the mesh between each step whilst preserving a suitable mesh quality that retains accuracy in the stress results. This significantly reduces the number of terms that need to be updated in the system matrix, thereby reducing the time required to carry out a re-analysis of the model. A range of solvers are assessed to find the most efficient and robust method of re-solving the system. The GMRES algorithm, using complete approximate LU preconditioning, is found to provide the fastest convergence rate.

Keywords: Boundary element method, Meshing, Re-analysis, Linear solvers

1. Introduction

Stress analysis of mechanical components has become an essential part of the validation of engineering designs but it can be time consuming and expensive. It is desirable to shorten the design cycle, thus reducing development costs and enabling new products to be brought to market in the shortest time possible. It is of particular importance to obtain the most suitable design at the conceptual design stage as the cost of any change at later stages grows rapidly. For many components this can be done only through rapid computational analysis of a wide range of initial geometries.

For clarity it is necessary to precisely define the terms real-time and rapid analysis within the context of this work. We use the definitions of Margetts *et al.* [1]. Real-time refers to the analysis taking place within the refresh rate of the media on which the model is viewed. This will typically be within 0.02 seconds. Rapid incorporates real-time but allows that the analysis may be slightly slower, taking place in an acceptably short period of time that the user is prepared to wait. This is of the order of seconds and is synonymous with the definition of interactive given in [1].

Real-time analysis of two-dimensional models is now a reality [2]. However, real-time stress analysis of three-dimensional objects presents numerous additional challenges. Over the last decade, various schemes based on the finite element method (FEM), that aim to provide interactivity have been presented [1, 3–5]. Meier *et al.* [6] review a range of deformable models that utilise both finite element (FE) and boundary element (BE) techniques. In this paper we address only BE implementations. The boundary element method (BEM) is a natural method to use where re-meshing is involved as changes need to be applied only to elements on the surface of the model. If the volume were meshed, as in the case of the FEM, then changes would propagate further through the model and many more degrees of freedom would be affected. Mackie [3] presents a substructuring approach that attempts to ameliorate these difficulties. Wang *et al.* [7] present a BEM based scheme for interactive analysis for surgical simulation. Real-time updating is also of interest to computer game developers to provide realistic deformable objects. For example, James and Pai [8] discuss the use of the BEM to provide physically accurate simulation of three-dimensional objects. However, as the technique is used purely for visual approximation of deformations, a much lower degree of accuracy is required and a coarse mesh can be used resulting in a very fast analysis. Pre-computed solutions are also utilised.

Concept Analyst [2], a two-dimensional BE stress analysis package that features real-time functionality, has already been developed. The current work aims to extend this interactive, real-time stress analysis capability into the three-dimensional domain. This involves the development of innovative techniques to generate, update and analyse the mesh. A key part of achieving this goal is the creation of a re-meshing algorithm. A high quality initial mesh must be created to produce an accurate ini-

*Corresponding author. Tel.: +44 191 3342487; fax: +44 191 3342408

Email addresses: timothy.foster@durham.ac.uk (T.M. Foster), m.s.mohamed@durham.ac.uk (M.S. Mohamed), jon.trevelyan@durham.ac.uk (J. Trevelyan), graham.coates@durham.ac.uk (G. Coates)

tial analysis using the minimum number of elements. The mesh should further be capable of absorbing some of the distortion introduced as the user updates the model. A concise data structure is also required to aid rapid re-meshing.

Many authors have proposed algorithms for re-meshing FE and BE models, often in an adaptive scheme. Adaptive refinement schemes, which are extensively reviewed for BEs by Kita and Kamiya [9], inform re-meshing through the errors in an initial analysis. Depending on the scheme selected different strategies may be employed. h-refinement requires subdivision of an existing mesh; p-refinement retains the existing mesh but alters the order of the element; r-refinement maintains the same number of elements but requires regeneration of the entire mesh. These schemes may be used in combination, but do not usually allow for geometrical changes. If the geometry of a model is changing, it is possible to refresh only affected areas of the mesh. This has previously been applied to fluid flow problems using the h-refinement technique with the FEM [10]. However, some of the methods employed could easily be adapted for BE analysis.

It is important to minimise the number of elements updated during re-meshing (Trevelyan *et al.* [11] have shown this to be the major factor in reducing re-analysis response time, as illustrated in Figure 1). Reducing the number of updated elements gives rise to a reduction in both the time to re-integrate and re-assemble the linear system of equations generated by the BEM and the time required to solve these equations [12]. Michler [13] uses techniques based on radial basis functions to locally distort a aircraft mesh based on geometric changes. This approach is designed for complex geometry undergoing significant geometric perturbation. A simpler and hence faster algorithm has been implemented for the basic geometries found in the current work. However, Michler's approach would be appropriate should more complex models be considered.

A fast linear solver, for example GMRES [14], can be employed to accelerate the re-analysis. Such iterative methods have already been applied to the BEM [15–17] and have produced positive results. However, they do not achieve the speed necessary for real-time re-analysis. As the majority of the mesh remains unchanged when the model geometry is updated, the results of the initial solve can be used to provide both a suitable initial guess at the solution and appropriate preconditioning of the evolving system matrix. Other linear solvers that utilise model order reduction have been developed by Leu [18], Amsallem and Farhat [19], Ryckelynck *et al.* [20] and Kerfriden *et al.* [21].

During dynamic updating of stress contours on continuously changing geometries, e.g. as a hole is dragged from one location to another, it is acceptable to relax requirements on the error associated with each individual simulation. Once the model is finalised a more refined mesh can be used to generate a more accurate estimation of the stress distribution.

The iterative solvers discussed in Section 6 were originally developed for application in the FEM. Here they have been applied to the BEM, which typically produces small, dense, non-symmetric matrices in contrast to the large, sparse systems found in the FEM.

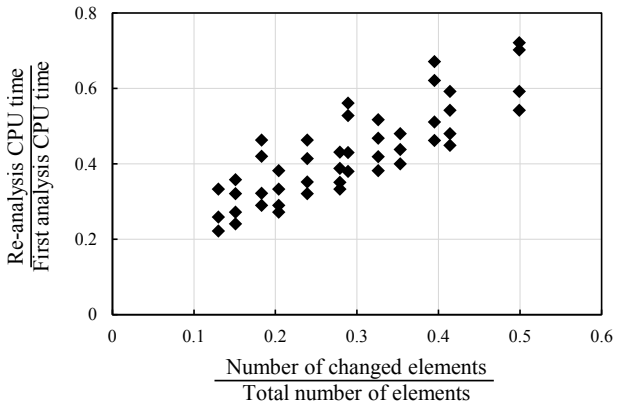


Figure 1: Re-analysis performance for three-dimensional applications (Reproduced from [11]).

This paper is organised as follows. In Section 2 the BEM is introduced. Section 3 describes the initial mesh, defining element and mesh quality measures. In Section 4 the re-meshing algorithm is outlined, and this is validated, based on stress results, in Section 5. In Section 6 six different re-analysis algorithms are discussed. These are assessed for numerical accuracy and speed in Section 7.

2. The Boundary Element Method

The BEM is a standard method of analysis in the solution of partial differential equations, and is the subject of numerous texts, including Becker [22]. This section contains a brief overview of the principal steps involved. We consider the problem of finding displacements and stresses in a linear elastic material comprising a domain $\Omega \in \mathbb{R}^3$, having boundary $\partial\Omega = \Gamma$. We seek to solve the equations of linear elasticity subject to boundary conditions

$$u(q) = \bar{u}, \qquad q \in \Gamma_u \tag{1}$$

$$t(q) = \bar{t}, \qquad q \in \Gamma_t \tag{2}$$

where u, t are displacement and traction components, $\bar{u}, \bar{t}$ are prescribed displacement and traction boundary conditions, and $\Gamma = \Gamma_u \bigcup \Gamma_t$. In practice, the use of different boundary condition types in different coordinate directions at the same location is common, so that this division of Γ into separate Neumann and Dirichlet boundaries in this fashion is purely symbolic. The boundary integral equation (BIE) can be formulated for displacements at a source point, $p \in \Gamma$, due to tractions and displacements on Γ.

$$c(p)u_j(p) + \int_\Gamma T_{ij}(p,q)u_i(q)d\Gamma(q) = \int_\Gamma U_{ij}(p,q)t_i(q)d\Gamma(q) \tag{3}$$

where $c(p)$ is a term introduced as a result of the limits applied to allow the strongly singular integral containing the traction kernel to be evaluated, so that the integral on the left hand side of (3) is evaluated in the Cauchy Principal Value sense. T_{ij} and

U_{ij} refer respectively to the traction and displacement kernels, given by:

$$T_{ij} = \frac{-1}{8\pi(1-\nu)r^2}\left[(1-2\nu)\delta_{ij} + 3\frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j}\right] + \frac{1-2\nu}{8\pi(1-\nu)r^2}\left[\frac{\partial r}{\partial x_j}n_i - \frac{\partial r}{\partial x_i}n_k\right] \quad (4)$$

$$U_{ij} = \frac{1}{16\pi(1-\nu)r}\left[(3-4\nu)\delta_{ij} + \frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j}\right] \quad (5)$$

where $r = |q - p|$, n_i and n_j are components of the outward normal, n, at q, δ_{ij} is the Kronecker delta, μ is the shear modulus and ν is the Poisson's ratio of the material. The subscripts define directional components, so that T_{ij} and U_{ij} refer to a traction or displacement in the Cartesian direction, i, at field point, q, caused by a unit load in direction j at the source point, p. To solve the system numerically the boundary of the object must first be discretized into a series of elements, forming a surface mesh. The discretized BIE can now be re-written in the local parametric element coordinates (ξ, η):

$$c(p)u_j(p) + \sum_{elem\ e}\int_{-1}^{1}\int_{-1}^{1} T_{ij}(p,q)\Phi_k(\xi,\eta)J(\xi,\eta)d\xi d\eta\ u_k = \sum_{elem\ e}\int_{-1}^{1}\int_{-1}^{1} U_{ij}(p,q)\Phi_k(\xi,\eta)J(\xi,\eta)d\xi d\eta\ t_k \quad (6)$$

where vector Φ contains the value of each shape function at the current integration point and J is the Jacobian, which transforms the differential variables at the current point from the local into the global coordinate systems. For reasons of computational performance we use the collocation form of the BEM, requiring collocation of (6) at a sufficient number of points, p, that for convenience coincide with the nodal positions. Performing the integrations given in (6) a set of equations can be derived. These are given in matrix form as:

$$[H]\{u\} = [G]\{t\} \quad (7)$$

where $[H]$ and $[G]$ contain the integrated traction and displacement kernels respectively. If an appropriate set of boundary conditions is defined, (7) can be rewritten in the form:

$$[A]\{x\} = \{b\} \quad (8)$$

This system can now be solved as a set of linear equations to find the unknown tractions and displacements contained in $\{x\}$. Internal stresses may be found by declaring p at the point of interest, substituting the now fully defined values of displacement and traction at the nodes into equation (6) and summing boundary integrals to yield $u(p)$.

3. Initial Mesh and Data Structure

A new mesh-generation algorithm has been developed for the current work. The algorithm ensures that sufficient control can be maintained over the mesh and data structure during the meshing and re-meshing procedures. The overheads are also reduced by generating only the data required for analysis and subsequent re-analysis. All the elements generated during meshing are continuous quadratic serendipity elements and may be either triangular or quadrilateral. Conical and cylindrical surfaces are meshed with a structured mesh whilst a constrained Delaunay triangulation scheme [23] is used to generate triangular meshes across plane surfaces. An example of a mesh generated is shown in Figure 2. Finer meshes can be generated as required according to user preferences.

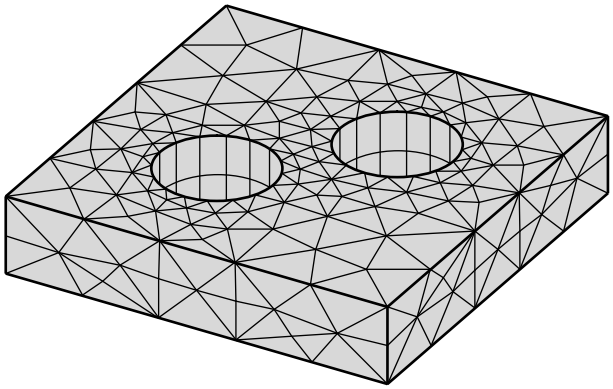


Figure 2: Three-dimensional mesh example.

The shape of the elements used in the mesh directly affects the accuracy of an integration scheme of prescribed order. The initial mesh is therefore generated from high quality quadratic BEs, for which the quality measure is described below, and is refined in areas where high stress concentrations have been predicted. The predictions are based on heuristics applied to the geometry of the initial model.

3.1. Element and mesh quality

A robust quality measure has been developed to ensure the initial mesh is of high quality and that an appropriate quality is maintained during the re-meshing procedure. The quality, Q, of each triangular element can be assessed using an adaption of the radius ratio discussed by Topping *et al.* [24]. This is defined as the ratio of the radius of the incircle to the circumcircle of the element. The quality, Q, can be formulated using this approach such that:

$$Q = \frac{16A^2}{abc(a+b+c)} \quad (9)$$

where A is the area of the element and a, b and c are the side lengths. Q takes a value from 0 and 1, where 1 indicates the highest quality element (an equilateral triangle) and 0 a fully collapsed element. It is prudent to ensure that every element is of quality, $Q \geq Q_{min}$. The mesh quality over each face of the model can be assessed using the mean, $\bar{Q}$, and standard deviation, S, of the collected element quality measures:

$$\bar{Q} = \frac{1}{n_E}\sum_{e=1}^{n_E} Q_e \quad (10)$$

$$S = \frac{1}{n_E}\sum_{e=1}^{n_E}\left(Q_e - \bar{Q}\right)^2 \quad (11)$$

where n_E is the number of elements on the assessed face. We define measures $\bar{Q}_{min}$ and S_{max} to be the minimum acceptable mean quality and the maximum acceptable standard deviation of element quality respectively.

Quadrilateral elements are only generated on cylindrical or conical surfaces and, using the procedures followed by the meshing and re-meshing algorithms, will always be of sufficient quality. If a measure were to be required the element aspect ratio could be used, along with some constraint on the internal angles.

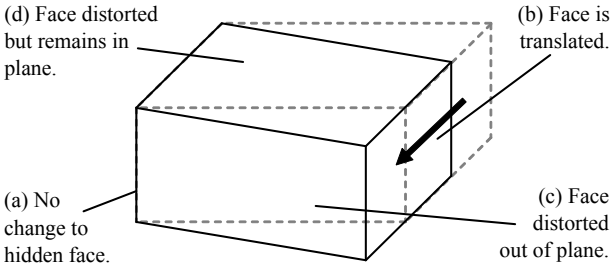


Figure 3: Geometric update of a block showing changed faces.

4. Re-meshing a geometric perturbation

When a geometric change is applied to a model the mesh must be updated. This is provoked by some dynamic cursor operation which is expected to be of pixel order. The mesh updating procedure is designed to minimise the number of elements affected when the model geometry is updated. This minimises the number of integrals, contained in the system matrix, that need to be recalculated before re-analysis and improves the convergence rate of a preconditioned iterative solver [12]. If the same total number of elements is maintained then the preconditioning matrix used by the solver can be re-used. If not then the matrix will need to be expanded. This could be done through block partitioning such as that described by Wang *et al.* [7]. For the purposes of this paper it has been assumed that the number of elements is constant. However the method could be expanded or improved by adding or removing elements.

During re-meshing, each face of the model is considered individually and the mesh updated according to how the geometry has been manipulated. Four situations can be identified and are shown in Figure 3:

(a) No change is made to the face: No change is made to the mesh.

(b) The face is translated: All the nodes and elements on the face are translated through the same vector. The integrals for the cases where point p and element e both lie on the face therefore do not need to be recalculated but other boundary integrals will need to be updated.

(c) The face is distorted out of plane: All elements on the face are updated. The mesh is regenerated across the face and all integrals will need to be recalculated.

(d) The face is distorted but remains in plane: Only some elements on the face require updating, the rest of the mesh remains unchanged.

We propose a new algorithm that can be applied to case (d) to propagate these changes beginning at updated edges around the face.

When the user updates the model, the changes propagate as follows: Nodes, N_i ($i = 1, 2, 3, ..., n_N$), distributed along updated edges are repositioned by translating or scaling the original distribution. The vector through which each has moved, V_i, is stored. Each element that includes N_i, E_{ei} ($e = 1, 2, 3, ..., n_{Ei}$), is assessed for quality, Q_{ei}, using (9). If $Q_{ei} < Q_{min}$, the unupdated corner node(s) of E_{ei} are moved through vector λV_i, where $\lambda \geq 1$. The coefficient λ is chosen such that the mesh changes do not propagate beyond this node for several future updates. The changes propagate further until all the elements return an acceptable value of Q_e. A node may be moved only once each time the model is re-meshed. This process is summarised in Figure 4.

The changes applied to the mesh often result in increasing numbers of updated and distorted elements and hence degradation of $\bar{Q}$ as demonstrated in Figure 5, which illustrates the case in which Q_{min} has been set to 0.5 to show the distortion more clearly. A high quality initial mesh absorbs some of the distortion but, to preserve accuracy, the mesh will periodically require more extensive modification over distorted faces. For large distortions, the mesh must be regenerated across the entire face (this rule has not been applied in Figure 5 so that the effects of failing to apply this measure can be seen). Once the updates have finished propagating, the mean and standard deviation of the element quality across face j, $\bar{Q}_j$ and S_j, are calculated. If $\bar{Q}_j < \bar{Q}_{min}$ or $S_j > S_{max}$, the mesh across face j is regenerated when the re-meshing algorithm is next called in place of the usual nodal updating procedure. For smaller problems, elements may be removed from areas where the mesh has become dense, such as in front of the advancing hole shown in Figure 5, and replaced in sparse areas, such as behind the hole. A smaller number of elements will be modified by this update although $\bar{Q}_j$ will not be substantially improved. Both techniques will result in an increase in run time for analyses where a part of the mesh has been regenerated but effectively reduce it on the following runs.

Figure 5 clearly shows that elements exist in a series of concentric bands around the hole. The same banding can be observed in almost all automatically generated meshes (FE or BE) around any geometric feature. Small perturbations in the location of the feature will affect only the neighbouring band. If the changes are larger, a cluster of elements of quality Q_{min} that move with the updated geometric feature will be maintained. Q_{min} may be graded by band to maintain a high element quality immediately around key geometry, leading to a more accurate approximation to the stress in these areas. However, maintaining a high element quality results in changes propagating further through the mesh and the usual compromise must be made between speed and accuracy.

The reader is reminded that, since the mesh is updated in

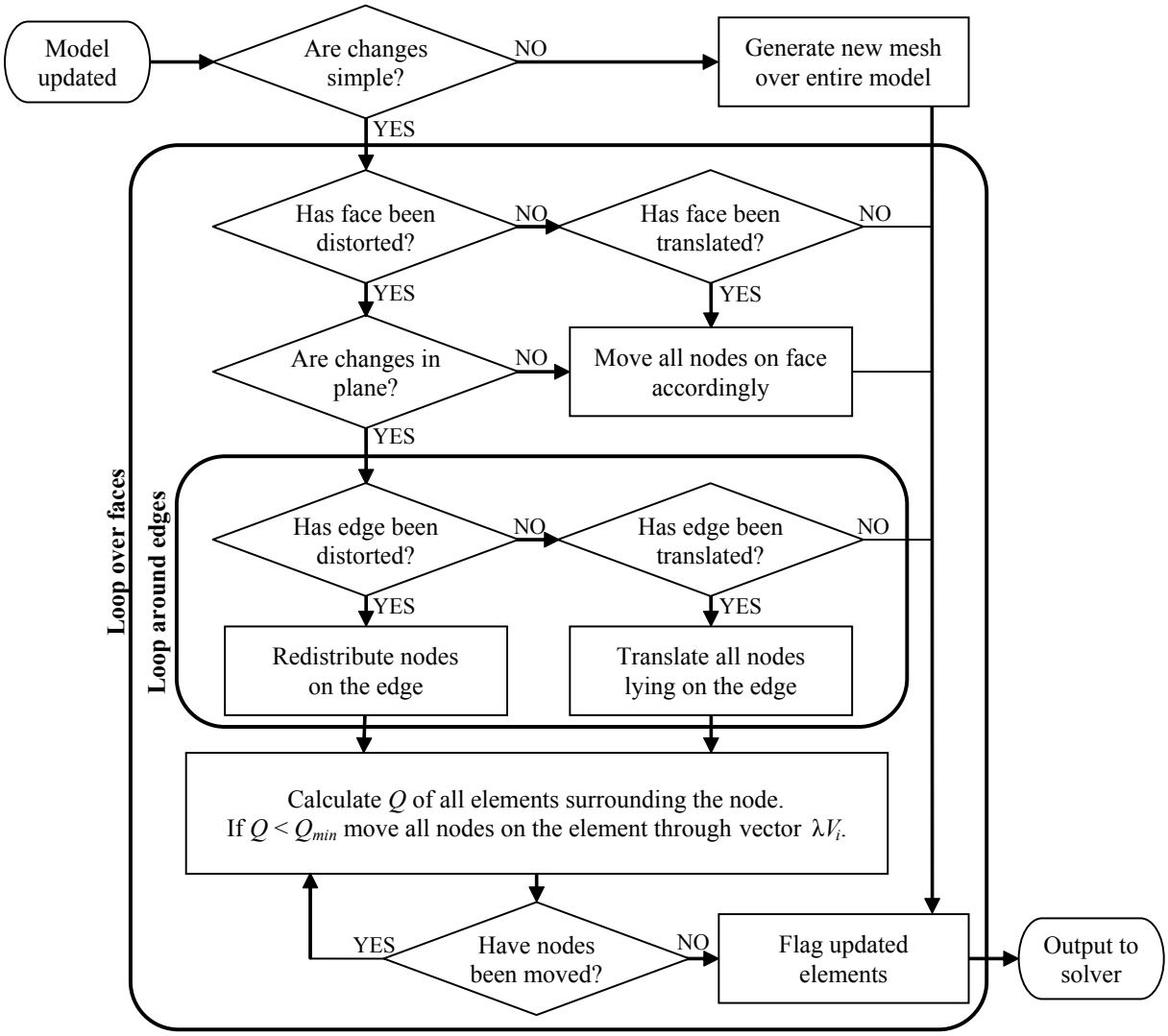


Figure 4: Flowchart showing a single re-meshing iteration.

real-time as the user drags features of the object into new locations, the majority of the geometric changes will be of pixel order and will require only minor modifications to the mesh. The larger mesh updates will be applied only in a very small proportion of the calls to the algorithm.

5. Validation of the re-meshing scheme

Tests have been carried out to validate the accuracy of stress solutions calculated using BE models perturbed by the re-meshing scheme. One example is discussed here, the case of moving a circular hole, of diameter D, within a thick plate of dimensions $15D \times 5D \times D$. The stresses at the nodes around the hole were computed using a BE mesh produced using the new algorithm and compared to benchmark stresses produced using a converged FE model. The FE model was converged to a L_2-norm tolerance of less than 1% in the stresses at the nodes around the hole. A long plate with a hole has been chosen to reduce the effects that the proximity of the ends of the plate have on the stresses around the hole. A single benchmark can therefore be used to compare the stresses for all hole locations as it is moved along the plate. The BE mesh on the surfaces $y = 0$ and $z = D$ of the model is shown in Figure 6. The following boundary conditions are applied:

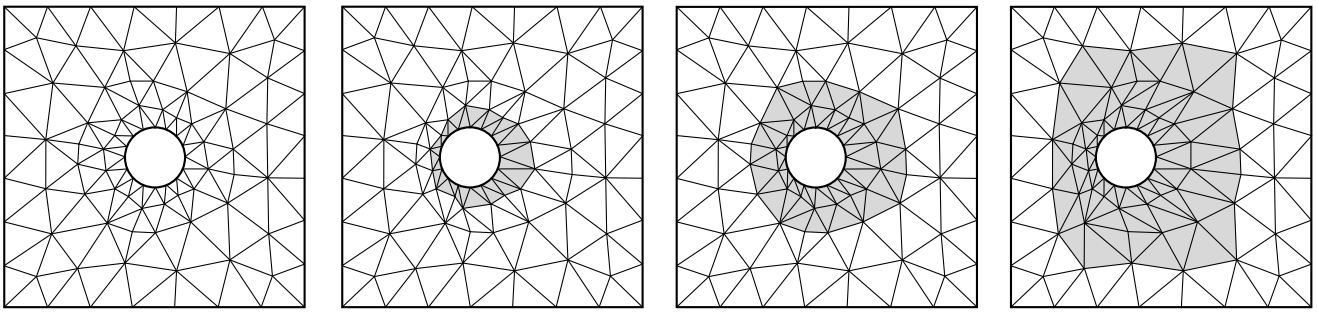
$$\begin{cases} u_x = 0, & x = 0 \\ u_y = 0, & y = 0 \\ u_z = 0, & z = 0 \\ t_x = T, & x = 15D \end{cases} \tag{12}$$

where u and t refer to displacements and tractions applied in the subscripted Cartesian direction. We take $T = 1000$.

An error measure, ε_σ, is calculated to compare the accuracy of the BE stresses to the benchmark:

$$\varepsilon_\sigma = \frac{\|\{\sigma\} - \{\hat{\sigma}\}\|}{\|\{\hat{\sigma}\}\|} \tag{13}$$

where $\|\cdot\|$ denotes the L_2-norm, $\{\hat{\sigma}\}$ the benchmark stress components and $\{\sigma\}$ the matching stress components generated by



(a) Initial mesh, $\bar{Q} = 0.89, S = 0.09$. (b) $\bar{Q} = 0.86, S = 0.12$. (c) $\bar{Q} = 0.82, S = 0.19$. (d) $\bar{Q} = 0.81, S = 0.17$.

Figure 5: Iterative updating of the mesh around a hole, showing updated elements. $Q_{min} = 0.5$.

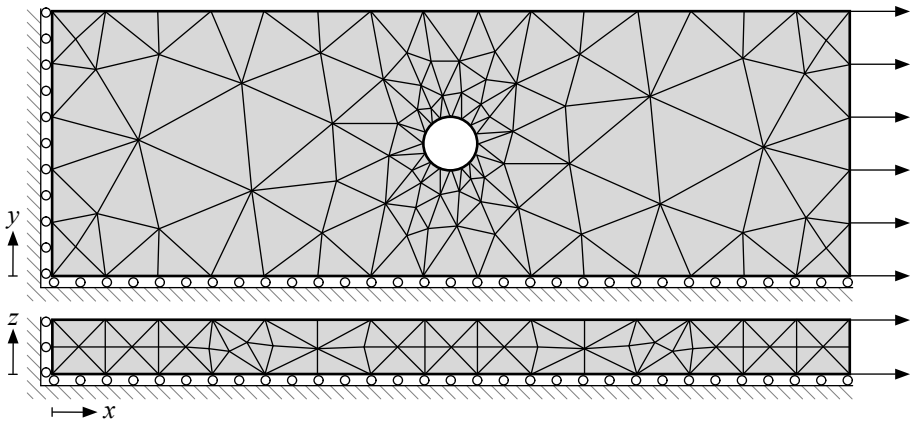


Figure 6: Test model.

the BE code. For this study, the tangential stresses, $\{\sigma_t\}$, around the top of the hole ($z = D$) will be used. Figure 7 shows the normalised tangential stress, σ_t/T, after the hole has moved a distance, $d = 0.5D$ in the positive x direction. Angle ϕ is defined as the anticlockwise angle from the top of the hole, as viewed in Figure 6. Note that, as this is a thick plate, the stress concentration factor, K_t, is smaller on the free surface of the plate than at $z = 0$ where $K_t \approx 3.1$ consistent with expectations for thin plate theory [25].

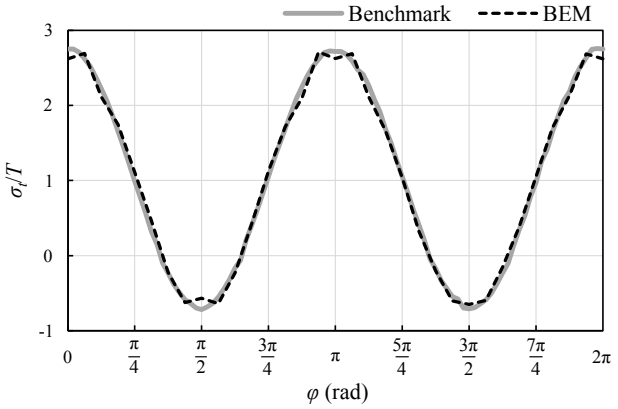


Figure 7: Normalised tangential stress, σ_t/T, around hole after it has moved through distance $D/2$.

The deviation in the BE results from the benchmark solution is most noticeable when $d\sigma_t/d\phi = 0$. Similar deviations also appear in the FE model if a comparable number of elements is used around the hole. All the elements around the hole were initially slightly distorted, having $\bar{Q} = 0.77$. However, at around $\phi = 3\pi/2$ rad; deformation of these elements as the mesh was updated resulted in a local increase in $\bar{Q}$ and hence produced a better approximation to $\hat{\sigma}_t$ when compared to $\phi = \pi/2$ where $\bar{Q}$ has degraded locally. A smoother and more accurate curve can be produced through appropriate application of higher quality elements; however, the trade-off between accuracy and computational resources required must be considered. The emphasis in the current work is to achieve very rapid solutions of acceptable quality during interactive re-analysis; higher quality solutions can be obtained using a more refined mesh. However, this will require a more computationally expensive analysis and should only be carried out to assess the final stress values.

A profile of how the model behaves as it is modified has been produced for the plate with a hole when it is meshed with 700 elements. The hole was moved through distance D in the positive x direction in 100 steps to simulate pixel order updates. The results are shown in Figures 8-11 where d is the total distance through which the hole has moved at any given point. It can clearly be seen in Figure 8 that if the hole is continuously moved in the same direction the number of updated elements will generally increase. To reduce this accumulation, local mesh regeneration must be carried out periodically us-

ing one of the schemes discussed in Section 4. Without these schemes, $\bar{Q}$ will, on average, decrease as the hole is moved further as shown in Figure 9, leading to an increase in ε_σ as shown in Figure 10. To show the effects of periodically regenerating the mesh an arbitrary value of $\bar{Q}_{min} = 0.84$ is selected, as shown by the grey lines in Figure 11, some of the element distortion is removed and $\bar{Q}$ is regulated. This in turn helps to regulate ε_σ. However, it is clear from Figure 10, which shows ε_σ without any regeneration, that $\Delta\varepsilon_\sigma$ remains small during re-meshing relative to ε_σ, accommodating some element distortion as evident in Figure 9. It should be noted that ε_σ is the L_2-norm error; the error in the peak stress is typically around $\varepsilon_\sigma/2$. Wall clock timings give the times for re-meshing in the region of thousandths of a second, around six times faster than generating the initial mesh.

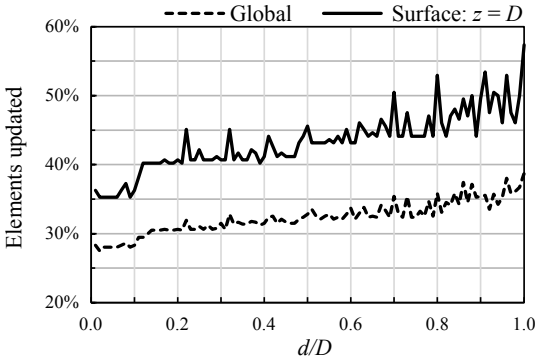


Figure 8: Percentage of elements updated for a hole in a plate.

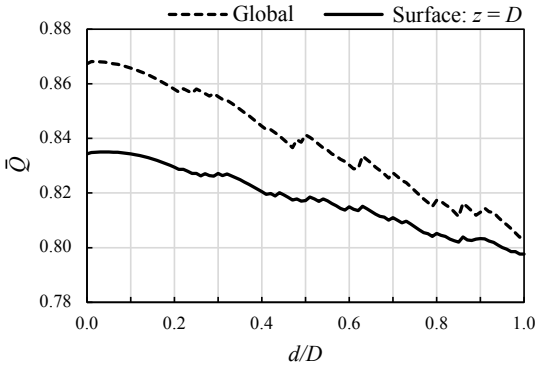


Figure 9: Mean element quality for a hole in a plate.

If applied intelligently a small number of high quality elements will produce a more accurate solution than many low quality elements. For the fastest analysis, it is therefore essential to find and generate an optimal mesh. Most stress concentrations peak around the edges of a model. If a high quality local mesh is maintained in these areas a greater accuracy can be achieved and maintained for these key results. However, this causes a greater number of elements to be updated in each iteration as the model is deformed and hence will increase the analysis time. A compromise must be reached to enable real-time updating whilst maintaining an acceptable accuracy.

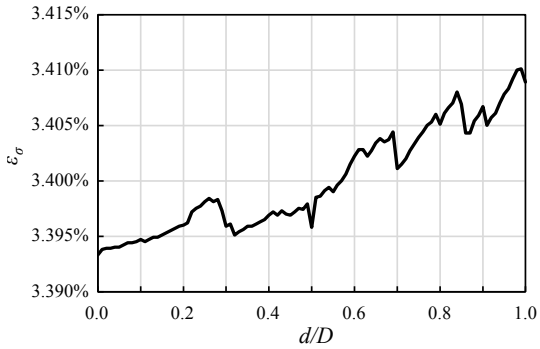


Figure 10: Error, ε_σ, at nodes around hole.

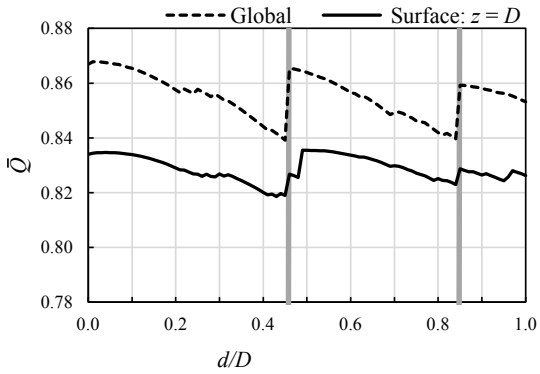


Figure 11: Mean element quality with regeneration.

6. Re-analysis of an updated model

For rapid analysis the model must be efficiently re-analysed as the geometry is updated. By re-writing (8) the system of n linear equations can be expressed as:

$$[A_i]\{x_i\} = \{b_i\} \tag{14}$$

where $i = 0, 1, 2, ...$ refers to the number of times the system has been modified. The initial system of equations, $i = 0$, generated from the initial geometry, must be solved before any dynamic updating of geometry occurs. This is carried out using a full LU decomposition. Every time the model is re-meshed the system is updated. As the majority of the changes are small the majority of matrix $[A_{i-1}]$ is preserved in $[A_i]$, where $[A_i] = [A_{i-1}] + [\Delta A]$. The matrix $[\Delta A]$ is sparse with only a few rows and columns containing non-zero values. The vector $\{b_i\}$ is entirely changed from $\{b_{i-1}\}$. The updated system must be re-solved to find the new tractions and displacements, $\{x_i\}$.

Two different approaches have been considered for rapidly re-solving the system; iterative solvers and reduction techniques. Six algorithms have been analytically compared for re-solving (14):

1. Generalised minimum residual (GMRES) [14]
2. Bi-Conjugate gradient stabilised (BiCGSTAB) [26]
3. Transpose free quasi-minimal residual (TFQMR) [27]
4. Leu reduction [18]

5. Eigenvector based proper orthogonal decomposition (POD) (E-POD) [20]

6. Singular value decomposition (SVD) based POD (SVD-POD) [21]

Algorithms 1-3 are iterative solvers whereas 4-6 utilise reduction techniques. The iterative algorithms and a full LU decomposition have also been tested on GPUs.

6.1. Iterative methods

Many different iterative solvers have been developed, the most common of which are discussed in [28]. The GMRES, BiCGSTAB and TFQMR algorithms have been chosen for this research as they are stable and applicable to non-symmetric systems. They are commonly applied to large sparse matrices typical of the FEM. Here they are applied to the smaller fully populated matrices produced by the BEM. The implementations are based on the templates given in [28] but have been modified to re-use as many vectors as possible to reduce data storage and memory accesses. It can be shown that if the data are spread over a larger area of memory the algorithms run more slowly. To reduce the amount of required memory the maximum number of iterations of each algorithm has been limited, enabling a suitable block of memory to be allocated beforehand. The modified pseudo-code for these algorithms can be found in the appendix.

A preconditioning matrix $[M]$ can be applied to (14) with the aim of speeding up convergence by improving the condition of the system that is to be solved. Both diagonal and complete approximate LU a [12] preconditioning have been applied to the GMRES, BiCGSTAB and TFQMR algorithms. The applied LU preconditioner is the complete LU decomposition of $[A_0]$, generated during the initial analysis. This is equivalent to applying $[A_0]^{-1}$ to (14) when a simple forward and backward substitution is carried out. For clarity this will be referred to simply as LU preconditioning throughout this paper. If it is assumed that each geometric update is small, $\{x_{i-1}\}$ can be used to reduce the solution time by providing a good first approximation of $\{x_i\}$ [12].

6.1.1. GMRES (Generalised minimal residual method)

The GMRES algorithm uses the Arnoldi method to generate a series of orthogonal vectors in the Krylov subspace, $\{v_j\}$ (where j is the iteration number) that can be used as a basis in which to construct the solution vector. Together with factors y_j, these are used to compute the solution:

$$\{x_i\} = \{x_{i-1}\} + y_1\{v_1\} + y_2\{v_2\} + ... + y_j\{v_j\} \tag{15}$$

or

$$\{x_i\} = \{x_{i-1}\} + [V]\{y\} \tag{16}$$

Convergence is reached when $\|\{r_j\}\|/\|\{b_i\}\| < tol$ where $\{r_j\}$ is the residual $\{b_i\} - [A_i]\{x_{ij}\}$ and $\|\cdot\|$ denotes the L_2-norm. The tolerance *tol* is defined by the user.

A new basis vector is generated with each iteration of the algorithm. Once the method has converged, (16) is applied to update $\{x_i\}$. To limit memory requirements, restarted versions of the GMRES method are often used to limit the number of Krylov vectors, updating $\{x_i\}$ before each restart. For the type of systems encountered in this work it has been found that the method converges within 20 iterations in the worst case. The majority of solutions take 5-10 iterations, therefore no restarts are required.

The amount of memory required by the solver, in addition to that required to store the system and preconditioner, increases with each iteration. For a typical 10 iteration solution, approximately an additional $12n$ double precision numbers need to be stored. This is small relative to the amount of memory occupied by the system and, for the sizes of system considered in the project, is far from prohibitive.

6.1.2. BiCGSTAB (Bi-conjugate gradient stabilised method)

The BiCGSTAB method was developed as an improvement on the conjugate gradient squared (CGS) method that often exhibits irregular convergence patterns. It attempts to minimise the residual vector, $\{r_j\}$:

$$\{r_j\} = Q_j([A_i])P_j([A_i])\{r_0\} \tag{17}$$

where j is the iteration number, $P_j([A_i])$ is an jth degree polynomial in $[A_i]$ and $Q_j([A_i])$ is an jth degree polynomial describing a steepest descent update. The vector $\{x_i\}$ is updated and convergence checked, using the same stopping criterion as the GMRES method, twice in each iteration. The BiCGSTAB method requires about half as much memory as the GMRES method at $6n$ double precision numbers.

6.1.3. TFQMR (Transpose free quasi-minimal residual method)

The quasi-minimal residual (QMR) method aims to solve the system in a least squares sense, in a similar manner to the GMRES approach. However, the basis generated in the Krylov subspace is bi-orthogonal and the residual is therefore referred to as quasi-minimal. The TFQMR algorithm achieves this without using $[A]^T$ which is slow to access due to the way a matrix is stored in computer memory. The total additional memory required by the TFQMR algorithm is similar to that of the BiCGSTAB algorithm at $7n$ double precision numbers.

The TFQMR algorithm has been expanded to reduce computation. This means that each loop around the algorithm is effectively two iterations. The vector $\{x_i\}$ is updated after each of these iterations, therefore only a single basis vector is required at any one time, thus reducing the memory requirements. The stopping condition computes an upper bound for the residual, $\{r_j\}$ using data already stored by the algorithm. This is compared to *tol*.

6.2. Reduction methods

Reduction techniques reduce the size of the problem by approximating it with a smaller system. This reduced system is very fast to solve but additional overheads are required to generate the system.

6.2.1. Leu

Leu [18] presents a reduction method, formulated specifically for the BEM, to solve (14). The theoretical speed up of the algorithm is computed relative to a direct method based on the number of floating point operations. On implementation the current authors found that the greater number of memory accesses required by the algorithm and the order in which the data must be retrieved from memory had a detrimental effect on the speed of the algorithm. It could therefore not compete with the other methods for the type of problem considered here. Due to these considerations the Leu algorithm will not be discussed further in this paper.

6.2.2. Proper orthogonal decomposition

Proper orthogonal decomposition (POD), also called Karhunen-Loève decomposition (KLD), uses a set of m basis vectors, $[B]$, together with m coefficients, $\{\zeta\}$, to find $\{x_i\}$:

$$\{x_i\} = [B]\{\zeta\} \tag{18}$$

The reduced system is constructed:

$$[B]^T[A_i][B]\{\zeta\} = [B]^T\{b_i\} \tag{19}$$

Here $[B]$ is of size $n \times m$ and $[A_i]$, $n \times n$, so that it remains only to solve a small $m \times m$ system where $m \ll n$. Hence $\{\zeta\}$ can be found with a direct solver.

Different approaches may be adopted for generating a suitable basis; two are presented here. Ryckelynck *et al.* [20] formulate a solution specifically for the BEM. The vectors $\{x_i\}$ from the first s analysis runs are used to construct matrix $[Q]$, which is used to produce matrix $[K]$:

$$[Q] = [\ \{x_0\}\quad \{x_1\}\quad ...\quad \{x_s\}\ ] \tag{20}$$

$$[K] = [Q][Q]^T \tag{21}$$

from which m eigenvectors are selected to form the basis, $[B]$, based on the related eigenvalues, α_k, where $\alpha_k > 10^{-10}\alpha_{max}$, $(k = 1, 2, 3, ..., n)$ and α_{max} is the highest eigenvalue. In the current article, this method has been denoted E-POD.

Kerfriden *et al.* [21] formulate their method for the FEM. They use a set of s vectors made up from a representative family of solutions to the initial model under differing initial conditions. These vectors are combined as in (20) to form $[Q]$ and the SVD is found:

$$[Q] = [U][S][V]^T \tag{22}$$

The first m columns of $[U]$ are taken to form an orthonormal basis $[B]$ where $m < s$. This method has herein been denoted SVD-POD.

6.3. GPU algorithms

No parallelisation has been applied in the CPU algorithms. However, preliminary work has been carried out into assessing the performance of some of the algorithms on graphics processing units (GPUs). The full LU decomposition, used in the initial solve, and the LU preconditioned GMRES, BiCGSTAB and TFQMR algorithms, used for the re-solve, have been coded for GPUs. This has been achieved using Nvidia's CUDA (Compute Unified Device Architecture) and the CUBLAS (CUDA Basic Linear Algebra Subprograms) library [29]. Small calculations have been retained on the CPU along with the Givens rotations used in the GMRES algorithm as these operations are not as suited to GPU parallelisation.

On account of the small number of calculations involved in re-solving the system using POD it would be counterproductive to run this solver on the GPU.

7. Assessment of solvers

To assess the speed and accuracy of the proposed solvers, three test models have been considered to cover a range of stress conditions:

1. Thick walled cylinder with an internal pressure (TWC). The internal radius has been reduced as the number of updates applied to the model, i, increases.

2. Cantilever under bending (CTL). The length of the cantilever has been extended as i increases.

3. Plate with a hole under uniaxial tension (PWH). This gives a more complex stress field than the uniform field found in the thick walled cylinder. The hole has been moved along the plate as i increases.

Each model has been analysed using several different meshes with varying degrees of refinement and different step sizes of geometric perturbation.

Error measures are generated by computing the L_2-norm of the difference between $\{x_i\}$ calculated using the iterative solver result and the benchmark solution, $\{\hat{x}_i\}$, based on a full LU decomposition. Note that only the error in the algebraic solution and not in the stress solution is considered. This error is computed using:

$$\varepsilon_x = \frac{||\{x_i\} - \{\hat{x}_i\}||}{||\{\hat{x}_i\}||} \tag{23}$$

Two types of test were carried out on the models. The iterative algorithms were timed for a fixed accuracy, $\varepsilon_x < 0.005\%$, and ε_x was compared for a fixed solve time of 0.05 seconds. The timings do not include the time required to evaluate the new boundary integrals. The decomposition methods are not iterative, therefore the accuracy cannot be prescribed in advance. These methods will also have a constant solve time for any given n.

The nomenclature used in the following sections will be as follows: the times, in seconds, to update and solve $[A_i]$ are denoted t_{ui} and t_{si} respectively. In the case $i = 0$, t_{u0} is the time to fully populate the system and t_{s0} is the time to solve the system using a full LU decomposition. When $i > 0$, t_{ui} is the time to update the system and t_{si} is the solve time of the appropriate solver. The number of updated degrees of freedom at the ith update is given by n_{ui}.

7.1. Repopulating the system

Repopulating the system involves recalculating the integrals stored in $[A_i]$. The relationship between n_{ui}/n and t_{ui}/t_{u0}, shown in Figure 12, is broadly linear although there is some variation between test models. No data was gathered for $23\% < n_{ui}/n < 40\%$. As the relationship is linear, reducing n_{ui} for any given model will proportionally reduce t_{ui}. This is in agreement with the findings of Trevelyan *et al.* [11], given in Figure 1 where the extra spread in re-analysis time is due to variability in t_{si}.

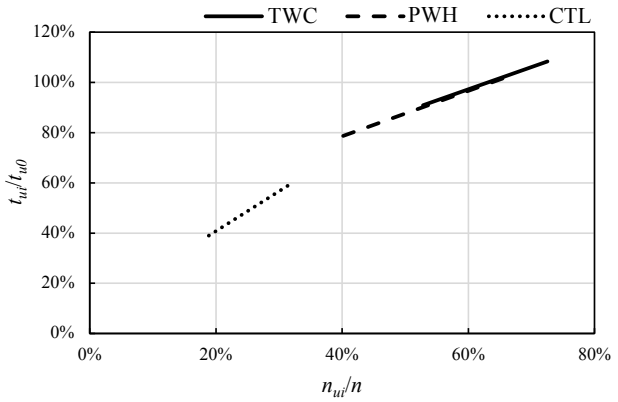


Figure 12: Repopulating the system.

Some overheads are involved in calculating which parts of the system to update. If a large percentage of the system is updated these can outweigh the benefits of only updating part of the system, as seen in Figure 12 when $n_{ui}/n > 60\%$.

7.2. Solution of the system for a fixed accuracy

The updated systems generated through geometric perturbation of a range of test models were solved using the iterative solvers and timed for a fixed accuracy, $\varepsilon_x < 0.005\%$. Figure 13 shows the relationship between t_{si} and n, for the diagonally and LU preconditioned iterative solvers after the first geometric update has been applied to each test model ($i = 1$). This relationship is of the form:

$$t_{si} = cn_{it}n^2 \tag{24}$$

The constant c depends on the solver used. The value n_{it} is dependent on the solver and the condition of the preconditioned system and is the number of iterations carried out by the solver. It should be noted that Figure 13 shows a typical fit to aid visualisation of the general trend. There is some variation in the actual data. For the diagonally preconditioned systems n_{it} is approximately constant for the test problems in this study, as seen in Figure 14, which shows t_{si} for the plate with a hole model discussed in Section 5. For the LU preconditioned systems the conditioning will initially degrade as the size of the perturbation from the original model geometry increases, causing n_{it} to increase. However once the geometry has changed significantly from the original, the degradation in the conditioning of the system will cease to substantially affect the solve time. This occurs at $d/D = 0.25$ in the case illustrated in Figure 14.

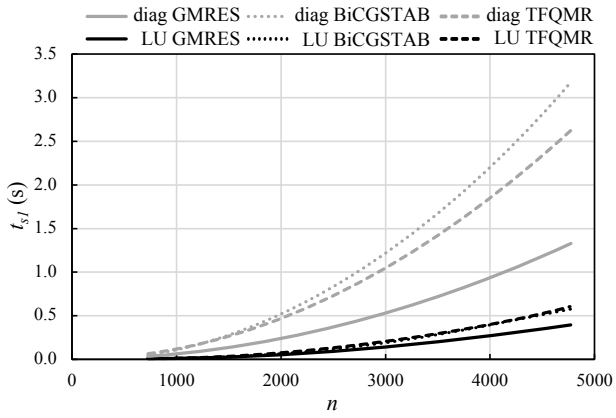


Figure 13: Comparison of iterative solve times for a fixed accuracy.

Where the time is available after the initial model is finalised and before re-analysis runs, a full LU decomposition should be computed since, when applied as a preconditioner, this can halve the re-analysis time for small geometric changes, as shown in Figure 14. However, if the model has small n, t_{si} will be more similar for the LU and diagonally preconditioned systems and, if a large modification is applied to a model with small n, the resulting system can be solved more quickly using diagonal preconditioning. This is due to the fact that in a small system a much higher percentage of the mesh will be regenerated after a geometric perturbation. The matrix, $[A_i]$, will therefore rapidly cease to bear much relation to $[A_0]$ and the LU and diagonal preconditioning will produce systems of comparable condition. This effect was only observed in the smallest ($n = 726$) of the 32 test cases assessed for this work and then only for the largest deformation. As the effect was negligible and $n > 726$ for the majority of models it can be safely assumed that LU preconditioning should be used to produce the smallest t_{si}.

If the total number of reanalysis updates, k is known and the aim is to reduce the total run time:

$$T = \sum_{i=1}^{k} t_{ui} + t_{si} \tag{25}$$

then diagonal preconditioning should be used in preference to LU preconditioning if $k < n/40$, as this will provide the same accuracy for a reduced T. During dynamic updating of the model k will not be known and, as the aim is to minimise $t_{ui}+t_{si}$, LU preconditioning should be applied. However, the LU preconditioner must be recomputed sparingly (if at all), as it is expensive to compute, and only after a minimum of $n/40$ updates have been applied to maintain the efficiency of the process. Alternatively an new LU preconditioner can be generated by a separate process in a new thread [12], thereby not detracting from the re-solution time.

Re-solve times and accuracies are model specific and different solvers may perform better under specific conditions but overall the GMRES approach most consistently provides most rapid convergence and has proved to be the most stable algorithm. Using these methods it appears that currently models of

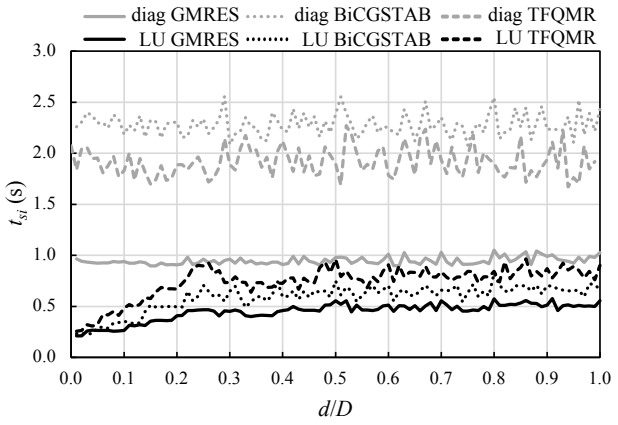


Figure 14: Resolve timings as a hole is moved along a thick plate ($n \approx 4300$).

size $n < 2000$ are amenable for real-time update of results.

7.3. Solution of the system within a predefined time

For rapid analysis, the solver will be required to re-solve the system to a sufficient accuracy within an acceptable time limit. To simulate this the iterative solvers have been allowed to run as many iterations as possible within 0.05 seconds. Reduction methods are not assessed here as the reduced problems always require a specific amount of overhead and are small enough to solve with a direct solver.

Figure 15 shows a diagrammatic representation of the error, ε_x, for a range of system sizes, n, based on numerical results. The contours depict the lower bound on ε_x for each solver. For LU preconditioned GMRES and BiCGSTAB solvers ε_x is consistently less than 10^{-3} (except in the occasional case). The error, ε_x, increases with n as larger systems require more time per iteration of the solver and therefore execute fewer iterations.

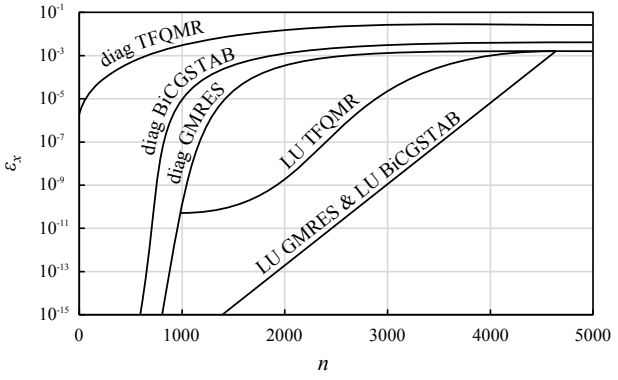


Figure 15: Comparison of iterative solve errors for a fixed solve time.

7.4. Solution of the system using POD

For the first set of tests, the basis, $[B]$, has been generated from the results of the first s analysis runs using both the E-POD and SVD-POD schemes. The re-solve time, t_{si}, is independent of $[B]$ and i and is of the form:

$$t_{si} = cmn^2 \tag{26}$$

where c is a constant. The variation in ε_x between the two schemes is typically of the order 10^{-6} when using the same m.

As shown in Figure 16, the time required to generate $[B]$, t_B, is large when applying E-POD as the eigenvalues of an $n \times n$ system must be computed. As t_B increases cubically with n, this is prohibitive for anything but the smallest models for this analysis application. When applying SVD-POD the SVD of an $n \times s$ system must be found; this leads to a considerably smaller t_B than E-POD. When computing the SVD-POD, t_B appears to bear little relationship to n for systems of the size encountered in this study ($n < 5000$) and is typically less than 0.1 seconds.

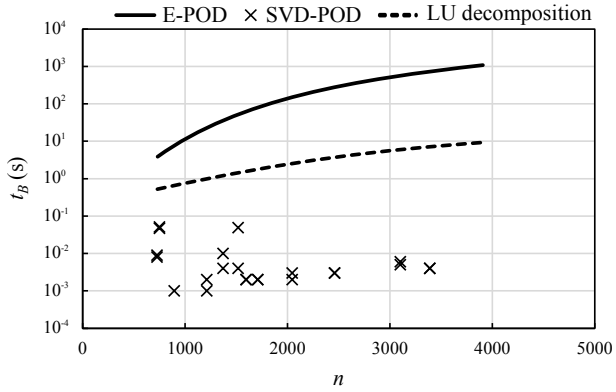


Figure 16: Time to generate $[B]$ for different system sizes.

System size, n, has little effect on ε_x which is mainly influenced by the number of basis vectors, m, and increases rapidly with i. The benefit of increasing m is limited, for example, in Figure 17, if $m > 3$ no reduction in the error is observed once $d/D = 0.08$. Both Kerfriden *et al.* [21] and Ryckelynck *et al.* [20] propose enrichment schemes to regulate ε_x by adding vectors to $[B]$ based on the latest analysis results. However, these enrichment schemes have not been implemented in the current work as any increase in t_{si} will render the method uncompetitive with the GMRES algorithm for the types of system encountered in this study.

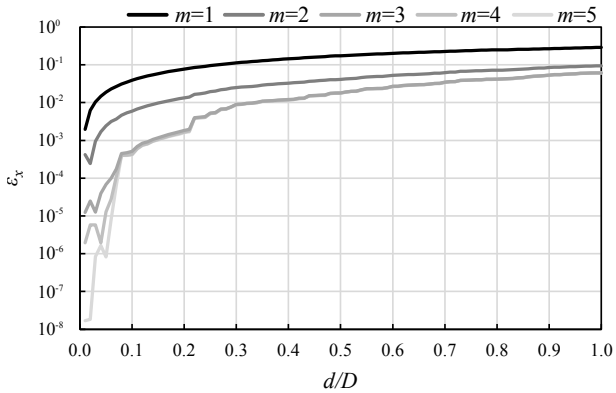


Figure 17: Change in solution error, ε_x, as a hole is moved diagonally across a thick plate.

Figure 18 has been produced from the test model shown in Figure 6 by moving the hole in the direction $x = y$. The error,

ε_x, is compared for the SVD-POD with $m = 3$ to ε_x for the LU and diagonally preconditioned GMRES algorithms. The same t_{si} has been applied to the GMRES algorithms as was used to solve the reduced system. It can be seen that, for this model, once $d/D = 0.08$, the GMRES methods provide a more accurate solution and ε_x does not degrade as rapidly. If a larger value of m were used these effects become even more pronounced.

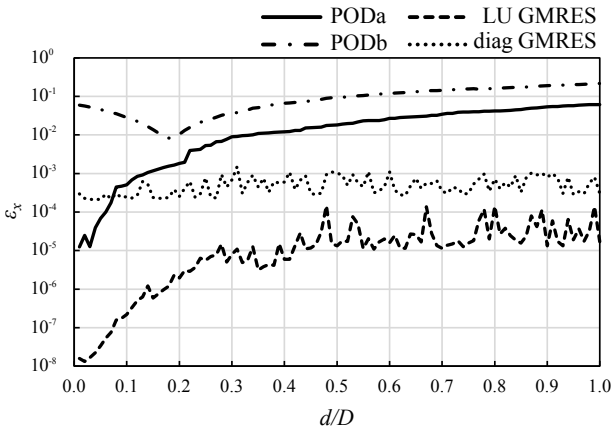


Figure 18: Solution error for a fixed solve time ($m = 3$).

In Figure 18, PODa denotes a set of runs using the first four analysis results to generate the basis, whereas PODb considers bases drawn from a set of representative solutions generated from the following models:

1. The initial model.
2. The hole was moved in the x-direction through distance $D/2$.
3. The hole was moved in the y-direction through distance $D/2$.
4. The plate was stretched in the x-direction through distance D.

PODb does not provide any improvement in ε_x over PODa. A poorly selected basis can also lead to an even greater ε_x.

7.5. Comparison of GPU and CPU implementations

Preliminary results for GPU implementations of the LU preconditioned GMRES, BiCGSTAB and TFQMR algorithms are given here. Figure 19 shows the mean ratio CPU time/GPU time, β, for the re-analysis solvers with a range of system sizes, n. The CPU results have been generated using an Intel Xenon X5570 CPU with the authors' solvers. The GPU results have been generated using an Nvidia Quadro 4800 GPU with the CUBLAS library. The relationship between β and n is approximately linear for systems of this size and it is found that it is only for systems of size $n > 5000$ that the GPU computation is beneficial. When generating and solving the LU decomposition used in the initial analysis, many more operations are applied to the same set of data and efficiency gains can be achieved for smaller systems of size $n > 2500$, as shown in Figure 20. The speed can be further improved if a blocked LU decomposition is used, where the block size, b, is chosen to fill the memory on the GPU. It is possible that using a similar blocking routine, such as an element-by-element form [30], with the iterative solvers could lead to a similar acceleration in these algorithms.

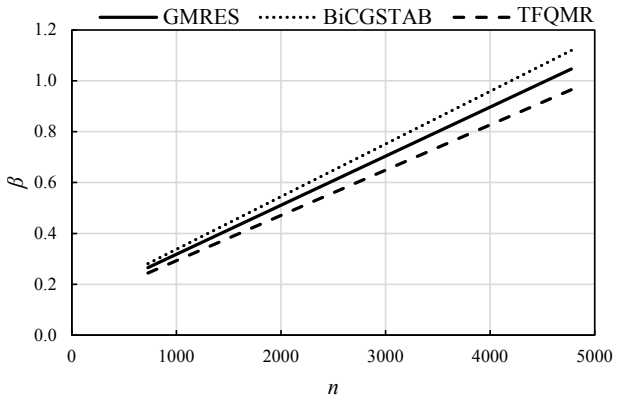


Figure 19: Comparison of CPU and GPU iterative solve times.

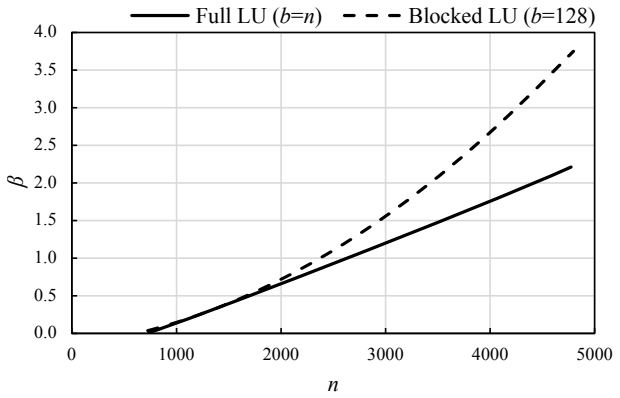


Figure 20: Comparison of CPU and GPU LU decomposition and solve times.

Some round off effects [31] were observed to affect the number of iterations in the BiCGSTAB solver only. In the worst case these had a negligible effect on the error, ε_x, of the order 10^{-8}. No variation in the solution accuracy was observed between the CPU and GPU results for the GMRES and BiCGSTAB solvers.

8. Conclusions

Progress has been made towards real-time three-dimensional BEM analysis. An algorithm has been introduced for updating a boundary element mesh to accommodate a geometric perturbation whilst preserving the majority of the mesh. The scheme is highly efficient for small perturbations; methods to improve the scheme for larger changes have also been discussed. Through careful control of propagation of re-meshing from updated geometry, a boundary element mesh can be updated in a computationally inexpensive manner so that the time required to update the boundary element system of equations can be reduced. An illustrative example has shown that this can be processed whilst maintaining an appropriate quality of solution.

Several linear solvers have been compared for solving this boundary element system of equations, with the complete approximate LU preconditioned GMRES method proving the most robust, outperforming other iterative and model order reduction based techniques. Using this method the speed of the re-solve can be improved by a factor of 100 when compared to a full LU decomposition. A GPU implementation of the GMRES, BiCGSTAB and TFQMR solvers has shown little to no speed up for the size of system assessed in this work. However, this should not be too readily dismissed as an option as GPU algorithms and hardware characteristics are currently developing and improving rapidly. Further optimisation work could lead to improved performance.

Future work will involve further reducing the time required for the computationally expensive process of re-population of the BEM system matrix terms and accelerating the solution further using adaptive cross approximation (ACA) [32].

Acknowledgements

This research is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) via grant EP/H000046/1. The authors would like to thank Dr. Stuart Spence of BAE Systems and Mr. Simon Walker of Jesmond Engineering for their input and support. GPUs have been supplied by Nvidia under the Academic Partnership Programme.

References

[1] L. Margetts, C. Smethurst, R. Ford, Interactive finite element analysis, in: NAFEMS World Congress, Malta, 2005.
[2] Concept Analyst Ltd., URL: www.conceptanalyst.com (2006).
[3] R. Mackie, An object-orientated approach to fully interactive finite element software, Advances in Engineering Software 29 (1998) 139–149.
[4] M. Ryken, J. Vance, Applying virtual reality techniques to the interactive stress analysis of a tractor lift arm, Finite Elements in Analysis and Design 35 (2000) 141–155.
[5] S. Terdalkar, J. Rencis, Graphically driven interactive finite element stress analysis for machine elements in the early design stage, Finite Elements in Analysis and Design 42 (2006) 884–899.
[6] U. Meier, O. López, C. Monserrat, M. Juan, M. Alcañiz, Real-time deformable models for surgery simulation: A survey, Computer Methods and Programs in Biomedicine 77 (2005) 183–197.
[7] P. Wang, A. Becker, I. Jones, A. Glover, S. Benford, C. Greenhalgh, M. Vloeberghs, Virtual reality simulation of surgery with haptic feedback based on the boundary element method, Computers and Structures 85 (2007) 331–339.
[8] D. James, D. Pai, ArtDefo - Accurate real time deformable objects, in: SIGGRAPH 99, 1999, pp. 65–72.
[9] E. Kita, N. Kamiya, Error estimation and adaptive mesh refinement in boundary element method, an overview, Engineering Analysis with Boundary Elements 25 (2001) 479–495.
[10] K. Yiu, D. Greaves, S. Cruz, A. Saalehi, A. Borthwick, Quadtree grid generation: Information handling, boundary fitting and CFD applications, Computers and Fluids 25 (8) (1996) 159–169.
[11] J. Trevelyan, P. Wang, S. Walker, A scheme for engineer-driven mechanical design improvement, Engineering Analysis with Boundary Elements 26 (2002) 425–433.
[12] J. Trevelyan, D. Scales, Techniques to accelerate BEM computation to provide virtual reality update of stress solutions, Engineering Analysis with Boundary Elements 31 (2007) 875–889.
[13] A. Michler, Aircraft control surface deflection using RBF-based mesh deformation, International Journal for Numerical Methods in Engineering 88 (2011) 986–1007.
[14] Y. Saad, M. Schultz, GMRES: A generalised minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific Computing 7 (1986) 856–869.
[15] K. G. Prasad, J. Kane, D. Keyes, C. Balakrishna, Preconditioned krylov solvers for bea, International Journal for Numerical Methods in Engineering 37 (1994) 16511672.
[16] F. Valente, H. Pina, Iterative techniques for 3-d boundary element method systems of equations, Engineering Analysis with Boundary Elements 25 (2001) 423–429.
[17] H. Xiao, Z. Chen, Numerical experiments of preconditioned Krylov subspace methods solving the dense non-symmetric systems arising from BEM, Engineering Analysis with Boundary Elements 31(12) (2007) 1013–1023.
[18] L. Leu, A reduction method for boundary element reanalysis, Computer Methods in Applied Mechanics and Engineering 178 (1999) 125–139.
[19] D. Amsallem, C. Farhat, An interpolation method for the adaptation of reduced-order models to parameter changes and its application to aeroelasticity, American Institute of Aeronautics and Astronautics Journal 46 (2008) 1803–1813.
[20] D. Ryckelynck, L. Hermanns, F. Chinesta, E. Alarćon, An efficient 'a priori' model reduction for boundary element models., Engineering Analysis with Boundary Elements 29 (2005) 796–801.
[21] P. Kerfriden, P. Gosselet, S. Adhikari, S. Bordas, Bridging proper orthogonal decomposition methods and augmented Newton-Krylov algorithms: An adaptive model order reduction for highly nonlinear mechanical problems, Computer Methods in Applied Mechanics and Engineering 200 (2011) 850–866.
[22] A. Becker, The Boundary Element Method in Engineering, McGraw-Hill International, 1992.
[23] T. Baker, Delaunay-Voronoi Methods, in: J.F. Thompson, B.K. Soni and N.P. Weatherill (Eds.), Handbook of Grid Generation, CRC Press, 1999.
[24] B. Topping, J. Muylle, P. Ivanyi, R. Putanowicz, B. Cheng, Finite Element Mesh Generation, Saxe-Coburg, 2004.
[25] W. Pilkey, Peterson's Stress Concentration Factors (Second Edition), John Wiley & Sons, 1997.
[26] H. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM Journal of Scientific and Statistical Computing 13 (1992) 631–644.
[27] R. Freund, A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems, SIAM Journal on Scientific Computing 14 (2) (1993) 470–482.
[28] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994.
[29] Nvidia, CUDA Toolkit 4.0 CUBLAS Library (2011).
[30] I. Smith, L. Margetts, G. Beer, C. Duenser, Parallelising the boundary element method using ParaFEM, in: Tenth International Symposium on Numerical Models in Geomechanics, Rhodes, Greece, Balkema, Netherlands, 2007.
[31] I. Smith, L. Margetts, The convergence variability of parallel iterative solvers, Engineering Computations 23(2) (2006) 154–165.
[32] M. Bebendorf, Approximation of boundary element matrices, Numerische Mathematik 86 (2000) 565–589.

Appendix: Iterative solver pseudo-code

The iterative solvers are based on the templates given in [28] with some modifications made by the authors. The nomenclature used in all the solvers is given in in Table 1 and the solver specific nomenclature in Table 2. Note that this may differ from the standard notation as some vectors have been reused to reduce the memory size and access requirements of the solvers.

Table 1: General Variables.

Symbol	Description
A	new system matrix
x_0, x	previous and current solution vectors
b	new b vector
r_0, r	initial and current residual vectors
M	preconditioning matrix
n	limit on number of solver iterations
tol	solution tolerance
$\|\| \cdot \|\|$	L_2-norm

Table 2: Solver specific variables.

	Variables	Vectors
GMRES*	$\alpha, \beta, \theta, \tau$	c, g, s, u, v, w, y
BiCGSTAB	$\alpha, \beta, \rho_1, \rho_2, \tau, \omega$	p, s, t, v
TFQMR	$\alpha, \beta, \gamma, \eta, \theta, \rho_1, \rho_2, \tau$	d, g, h, u, v, w

*GMRES incorporates matrices H and V, where V contains vectors, v.

Algorithm 1 GMRES

$r_0 = b - Ax_0$
$\tau = b^T b$
$g_0 = \|r_0\|$
$v_0 = r_0/g_0$
$i = 0$
while $i < n$ **do**
  Solve: $Mw = Av_i$
  for $j = 0, 1, 2, ..., i$ **do**
    $H_{i,j} = w^T v_j$
    $w = w - H_{i,j} v_j$
  end for
  $v_{i+1} = w/\|w\|$
  for $j = 0, 1, 2, ..., i$ **do**
    $\alpha = H_{i,j} c_j - H_{i,j+1} s_j$
    $\beta = H_{i,j} s_j - H_{i,j+1} c_j$
    $H_{i,j} = \alpha$
    $H_{i,j+1} = \beta$
  end for
  $\theta = -\tan^{-1}(\|w\|/H_{i,i})$
  $c_i = \cos\theta$
  $s_i = \sin\theta$
  $H_{i,i} = H_{i,i} c_i - \|w\| s_i$
  $\alpha = g_i c_i$
  $\beta = g_i s_i$
  $g_i = \alpha$
  $g_{i+1} = \beta$
  if abs$(\beta)/\tau < tol$ **then**
    Converged: end while
  end if
  $i = i + 1$
end while
Solve: $H^T y = g$
$u = V^T y$
Solve: $Mw = u$
$x = x_0 + w$

Algorithm 2 BiCGSTAB

$r_0 = b - Ax_0$
$\tau = \|b\|$
$r = r_0$
$x = x_0$
$i = 0$
while $i < n$ **do**
  $\rho_1 = r_0^T r$
  if $i = 0$ **then**
    $p = r$
  else
    $\beta = (\rho_1/\rho_2)(\alpha/\omega)$
    $p = r + \beta(p - \omega v)$
  end if
  Solve: $Ms = p$
  $v = As$
  $\alpha = \rho_1 / r_0^T v$
  $x = x + \alpha s$
  $r = r - \alpha v$
  if $\|r\|/\tau < tol$ **then**
    Converged: end while
  end if
  Solve: $Ms = r$
  $t = As$
  $\omega = t^T r / t^T t$
  $x = x + \omega s$
  $r = r - \omega t$
  if $\|r\|/\tau < tol$ **then**
    Converged: end while
  end if
  $\rho_2 = \rho_1$
  $i = i + 1$
end while

Algorithm 3 TFQMR

$Mr = b - Ax_0$
$Mv = Ar$
$g = v$
$w = u = r$
$\rho_1 = r^T r$
$\tau = \sqrt{\rho_1}$
$x = x_0$
$i = 0$
while $i < n$ **do**
  $\alpha = \rho_1 / r^T v$
  if $i = 0$ **then**
    $d = u$
  else
    $\beta = \theta^2 \eta / \alpha$
    $d = u + \beta d$
  end if
  $w = w - \alpha g$
  $\theta = \|w\|/\tau$
  $\gamma = 1/(1 + \theta^2)$
  $\eta = \gamma \alpha$
  $\tau = \tau\theta\sqrt{\gamma}$
  $x = x + \eta d$
  $i = i + 1$
  if $\tau\sqrt{i} < tol$ **then**
    Converged: end while
  end if
  $u = u - \alpha v$
  Solve: $Mh = Au$
  $\beta = \theta^2 \gamma$
  $d = u + \beta d$
  $w = w - \alpha h$
  $\theta = \|w\|/\tau$
  $\gamma = 1/(1 + \theta^2)$
  $\eta = \gamma \alpha$
  $\tau = \tau\theta\sqrt{\gamma}$
  $x = x + \eta d$
  $i = i + 1$
  if $\tau\sqrt{i} < tol$ **then**
    Converged: end while
  end if
  $\rho_2 = r^T w$
  $\alpha = \rho_2 / \rho_1$
  $u = w + \alpha u$
  Solve: $Mg = Au$
  $v = g + \alpha(h + \alpha v)$
  $\rho_1 = \rho_2$
end while